Information Sciences 178 (2008) 2985–2999

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Large scale evolutionary optimization using cooperative coevolution ☆

Zhenyu Yang [a], Ke Tang [a,*], Xin Yao [a,b]

[a] Nature Inspired Computation and Applications Laboratory (NICAL), Department of Computer Science and Technology, University of Science and Technology of China, Hefei, Anhui 230027, China

[b] The Centre of Excellence for Research in Computational Intelligence and Applications (CERCIA), School of Computer Science, The University of Birmingham, Edgbaston, Birmingham B15 2TT, UK

ARTICLE INFO

Article history:
Received 10 June 2007
Received in revised form 14 November 2007
Accepted 25 February 2008

Keywords:
Global optimization
High-dimensional optimization
Evolutionary algorithms
Cooperative coevolution
Differential evolution

ABSTRACT

Evolutionary algorithms (EAs) have been applied with success to many numerical and combinatorial optimization problems in recent years. However, they often lose their effectiveness and advantages when applied to large and complex problems, e.g., those with high dimensions. Although cooperative coevolution has been proposed as a promising framework for tackling high-dimensional optimization problems, only limited studies were reported by decomposing a high-dimensional problem into single variables (dimensions). Such methods of decomposition often failed to solve nonseparable problems, for which tight interactions exist among different decision variables. In this paper, we propose a new cooperative coevolution framework that is capable of optimizing large scale nonseparable problems. A random grouping scheme and adaptive weighting are introduced in problem decomposition and coevolution. Instead of conventional evolutionary algorithms, a novel differential evolution algorithm is adopted. Theoretical analysis is presented in this paper to show why and how the new framework can be effective for optimizing large nonseparable problems. Extensive computational studies are also carried out to evaluate the performance of newly proposed algorithm on a large number of benchmark functions with up to 1000 dimensions. The results show clearly that our framework and algorithm are effective as well as efficient for large scale evolutionary optimisation problems. We are unaware of any other evolutionary algorithms that can optimize 1000-dimension nonseparable problems as effectively and efficiently as we have done.

1. Introduction

Evolutionary optimization has achieved great success on many numerical and combinatorial optimization problems in recent years [14]. However, most evolutionary algorithms (EAs) still suffer from the "curse of dimensionality", which implies that their performance deteriorates rapidly as the dimensionality of the search space increases [22]. EAs that perform well on low-dimension problems often fail to find good near optimal solutions to high-dimensional problems. Some initial efforts in tackling large high-dimensional problems made unrealistic assumptions that the problem is separable. This paper will focus on nonseparable functions since separable functions can easily be decomposed into small subproblems and are not as high-dimensional as they appear.

☆ Part of this work was published in the Proceedings of the 2007 IEEE Congress on Evolutionary Computation [25].

* Corresponding author.
E-mail addresses: zhyuyang@mail.ustc.edu.cn (Z. Yang), ketang@ustc.edu.cn (K. Tang), x.yao@cs.bham.ac.uk (X. Yao).

0020-0255/$ - see front matter © 2008 Elsevier Inc. All rights reserved.
doi:10.1016/j.ins.2008.02.017

Cooperative coevolution [9,6] has been proposed to solve large and complex problems through problem decomposition. It can be regarded as an automatic approach to implementing the divide-and-conquer strategy. The original framework of cooperative coevolution (CC) [10] for optimisation can be summarised as follows:

Problem decomposition: Decompose a high-dimensional objective vector into smaller subcomponents that can be handled by conventional EAs.
Subcomponent optimization: Evolve each subcomponent separately using a certain EA.
Subcomponents coadaptation: Since interdependencies may exist between subcomponents, coadaptation (i.e., coevolution) is essential in capturing such interdependencies during optimization.

A critical step in the above framework is problem decomposition [11]. An ideal CC algorithm should decompose a large problem into subcomponents where the interdependencies among different subcomponents are minimal. Existing algorithms [6,10,15,16] originated from the above CC framework used two simple problem decomposition methods, i.e., the one-dimensional based and splitting-in-half strategies. The one-dimensional based strategy decomposes a high-dimensional vector into single variables. In other words, an n-dimensional problem would be decomposed into n one-dimensional problems. While this strategy is very simple and effective for separable functions, it did not consider interdependencies among variables for nonseparable problems. As a result, it is unable to tackle nonseparable problems satisfactorily. The splitting-in-half strategy always decompose a high-dimensional vector into two equal halves and thus reducing an n-dimensional problem into two $\frac{n}{2}$-dimensional problems. If n is large, the $\frac{n}{2}$-dimensional problems would still be very large and challenging to solve. Although one might be able to apply the splitting-in-half strategy recursively, no work has been reported so far. It is unclear when and how interdependencies among different variables can be captured for nonseparable problems.

In addition to the lack of explicit consideration of variable interdependencies for nonseparable problems, existing CC-based algorithms still use outdated EAs as the subcomponent optimizers, while more effective EAs have been proposed in recent years. The performance of CC optimization algorithms on high-dimensional problems can be enhanced significantly by adopting more advanced EAs.

With the exception of FEPCC [6], the existing CC-based algorithms were applied to problems with only up to 100 dimensions, which are still relatively small for many real-world problems. No work has been reported in the literature for any EAs to tackle nonseparable problems of up to 1000 dimensions.

This paper introduces a new problem decomposition strategy, i.e., the grouping based strategy, in order to better capture the variable interdependencies for nonseparable problems [25]. A simple yet effective decomposition methods using this strategy is presented. An adaptive weighting strategy is also introduced in order to strengthen further coadaptation among decomposed subcomponents when they are interdependent. The idea behind our new algorithm is to optimize a group of interacting variables together (as a subcomponent), rather than splitting them into different subcomponents. Ideally, each subcomponent should consist of tightly interacting variables while the interaction among subcomponents should be weak. Our grouping and adaptive weighting strategies are attempts towards this direction.

In addition to the decomposition strategies, this paper also introduces a new differential evolution (DE) variant, SaNSDE, as the base optimizer for subcomponents. The key feature of SaNSDE is incorporation of self-adaptive neighbourhood search into DE. SaNSDE has been shown to perform very well, without any coevolution, on a large number of benchmark functions. The results in this paper shows further that SaNSDE can also help coevolutionary optimization significantly. In fact, our new algorithm is able to tackle nonseparable problems with up to 1000 dimensions.

Although our new algorithms have shown superior performance on many benchmark functions, our decomposition strategies are still rather simple and can be improved further. Extensive analyses and computational studies have been carried out and reported in this paper, which explain why, how and when our new algorithms work well.

The rest of this paper is organized as follows. Section 2 gives a brief review of cooperative coevolution. Section 3 presents our new CC framework and analyzes its effectiveness. Our grouping and adaptive weighting strategies will be described. Theoretical analysis is given to explain why and how such simple strategies can work well. Section 4 introduces the DE with self-adaptive neighbourhood search — SaNSDE. Section 5 describes our computational studies and presents the experimental results on a diverse set of nonseparable functions with up to 1000 dimensions. Finally, Section 6 concludes this paper with some remarks and future research directions.

2. Cooperative coevolution

"As evolutionary algorithms are applied to the solution of increasingly complex systems, explicit notions of modularity must be introduced to provide reasonable opportunities for solutions to evolve in the form of interacting coadapted subcomponents" [9]. Examples of this show up in the need for rule hierarchies in classifier systems and subroutines in genetic programming [10]. CC is a general framework for applying EAs to large and complex problems using a divide-and-conquer strategy. In CC, the objective system (such as a vector) is decomposed into smaller modules and each of them is assigned to a species (i.e. subpopulation). The species are evolved mostly separately with the only cooperation happening during fitness evaluation.

The original cooperative coevolution (CC) framework for high-dimensional optimization can be summarized as follows [6,10,15,16]:

(1) Decompose an objective vector into m low-dimensional subcomponents.
(2) Set $i = 1$ to start a new *cycle*.
(3) Optimize the ith subcomponent with a certain EA for a predefined number of fitness evaluations (FEs).
(4) If $i < m$ then $i++$, and go to Step 3.
(5) Stop if halting criteria are satisfied; otherwise go to Step 2 for the next *cycle*.

Here a *cycle* consists of one complete evolution of all subcomponents. The main idea is to decompose a high-dimensional problem into some low-dimensional subcomponents and evolve these subcomponents cooperatively for a predefined number of *cycles*. The cooperation occurs only during fitness evaluation. The size of each subcomponent should be within the optimization ability of the EA used.

Potter applied CC to concept learning and neural network construction [9]. García-Pedrajas et al. proposed COVNET, which is a new CC model for evolving artificial neural networks [4]. In the domain of learning multiagent behaviors, CC represents a natural approach to applying traditional evolutionary computation [7,8]. Besides, CC has also been employed in real-world applications, e.g. Cao et al. adopt a CC-based approach in their pedestrian detection system [2]. High-dimensional optimization is another appropriate application of CC. Several CC EAs have been proposed to optimize high-dimensional problems [6,10,15,16]. These algorithms were very successful in optimizing problems with independent variables. These ability in dealing with nonseparable problems were somewhat limited.

As pointed out in Section 1, existing CC algorithms for optimization suffer from three major shortcomings. First, the decomposition strategies did not take into account variable interdependencies for nonseparable problems. Second, the base optimizer (e.g., an EA) was out of date. Third, the CC algorithms were able to deal with problems with only up to 100 dimensions.

3. The new framework with grouping and adaptive weighting

We propose a new CC framework with a group-based problem decomposition strategy. In addition, the new CC framework is designed to change grouping structures dynamically, which will increase the chance of optimizing interacting variables together (and thus more effectively). Moreover, we maintain coadaptation among different groups for overall controlling. More details are given below.

The crucial idea of our new group-based framework is to split an objective vector into m s-dimensional subcomponents (assuming $n = m^*s$), and then evolve each of them with an EA. For coadaptation, we propose an adaptive weighting strategy, which applies a weight to each of these subcomponents after each cycle, and evolves the weight vector with a certain optimizer. The key steps of the framework can be summarized as follows:

(1) Set $i = 1$ to start a new *cycle*.
(2) Split an n-dimensional object vector into m subcomponents (s-dimensional) randomly, i.e. $n = m^*s$. Here "randomly" means that each variable has the same chance to be assigned into any of the subcomponents.
(3) Optimize the ith subcomponent with a certain EA for a predefined number of FEs.
(4) If $i < m$ then $i++$, and go to Step 3.
(5) Apply a weight to each of the subcomponents. Evolve the weight vectors for the best, the worst and a random members of current population.
(6) Stop if halting criteria are satisfied; otherwise go to Step 1 for the next *cycle*.

The main differences between our proposed approach and the existing ones (e.g., [10,6]) are: (1) our new framework evolves a group of variables together, and the grouping structure will be changed dynamically; (2) the new framework uses adaptive weighting for coadaptation among subcomponents after each cycle. This scheme of algorithm will be denoted as EACC-G in our paper.

The motivation behind the adaptive weighting strategy is to provide coadaptation over all subcomponents when they are interdependent. After each cycle, we apply a weight to each of these subcomponents. These weights will build up a weight vector. For any member of a population, there exists an optimal weight vector. The determination of the optimal weight vector is an optimization problem itself. Fortunately, this optimization problem has a much lower dimensionality than the original one, and thus can usually be handled by existing EAs. Why and how the adaptive weighting strategy works is further demonstrated as follows:

(1) For any individual, $\mathbf{x} = (x_1, x_2, \ldots, x_n)$, of a population, it is true that:

$$f(\mathbf{x}) \equiv f(\mathbf{w_c} \cdot \mathbf{x})$$

where $\mathbf{w_c} = (1, 1, \ldots, 1)$ indicates a constant weight vector.

(2) To obtain better fitness value, we can apply a weight w_i to each component of $\mathbf{x}$, and then optimize the weight vector. So we achieve:

$$\min_{\mathbf{w}} f(\mathbf{w}\cdot\mathbf{x}) \leqslant f(\mathbf{w_c}\cdot\mathbf{x}) \equiv f(\mathbf{x})$$

where $\mathbf{w} = (w_1, w_2, \ldots, w_n)$ is the weight vector over the individual $\mathbf{x}$.

(3) However, optimizing the weight vector $\mathbf{w}$ is as hard as to optimize the original individual $\mathbf{x}$, since they are in the same dimension. But DECC-G splits the n-dimensional vector $\mathbf{x}$ into m ($m \ll n$) subcomponents. So we can alternatively apply a weight to each of these subcomponents, and thus we only need to optimize a much lower dimensional vector $\tilde{\mathbf{w}} = (\tilde{w}_1, \tilde{w}_2, \ldots, \tilde{w}_m)$:

$$\min_{\mathbf{w}} f(\mathbf{w}\cdot\mathbf{x}) \leqslant \min_{\mathbf{w}'} f(\mathbf{w}'\cdot\mathbf{x}) \leqslant f(\mathbf{x})$$

where $\mathbf{w}' = (\underbrace{\tilde{w}_1, \ldots, \tilde{w}_1}_{s}, \underbrace{\tilde{w}_2, \ldots, \tilde{w}_2}_{s}, \ldots, \underbrace{\tilde{w}_m, \ldots, \tilde{w}_m}_{s})$, with s denotes the dimension of each subcomponent and m denotes the number of subcomponents (assuming $n = m^* s$).

So the adaptive weighting strategy works like a trade off between optimizing high-dimensional vector $\mathbf{w}$ and no weighting at all. Further, since the variables of a subcomponent is controlled integrally by changing the weight of it, the process of optimizing the weight vector can also be viewed as a coarse coadaptation over all subcomponents.

3.1. Why and How well EACC-G works

In spite of many discussions of separable and nonseparable functions in the literature, there is no clear definition to distinguish between separable and nonseparable problems. Based on the description in a recent suite of benchmark functions [19], we provide the following definition in this paper.

Definition 1. $f(\mathbf{x})$ is called a separable function if $\forall k \in \{1, n\}$ and

$$\left.\begin{array}{l} \mathbf{x} \in S, \quad \mathbf{x} = (\mathbf{x_1}, \ldots, \mathbf{x_k}, \ldots, \mathbf{x_n}) \\ \mathbf{x}' \in S, \quad \mathbf{x}' = (\mathbf{x_1}, \ldots, \mathbf{x}'_{\mathbf{k}}, \ldots, \mathbf{x_n}) \end{array}\right\} \Rightarrow f(\mathbf{x}) < f(\mathbf{x}') \tag{1}$$

imply

$$\left.\begin{array}{l} \forall \mathbf{y} \in S, \quad \mathbf{y} = (\mathbf{y_1}, \ldots, \mathbf{x_k}, \ldots, \mathbf{y_n}) \\ \forall \mathbf{y}' \in S, \quad \mathbf{y}' = (\mathbf{y_1}, \ldots, \mathbf{x}'_{\mathbf{k}}, \ldots, \mathbf{y_n}) \end{array}\right\} \Rightarrow f(\mathbf{y}) < f(\mathbf{y}') \tag{2}$$

Otherwise, $f(\mathbf{x})$ is called a nonseparable function.

Basically, non-separability means that the objective vector consists of interacting variables, while separability means that the influence of a variable on the fitness value depends only on itself, i.e., independent of any other variables.

Given a nonseparable function, if all of its variables are highly interdependent of each other, no CC algorithms would perform well on such an extreme case. For many real-world problems, interdependencies often occur among subsets, but not all, of variables. For such problems, our grouping based strategies can be an effective approach to decompose a high-dimensional vector into smaller low-dimensional subcomponents. Within each subcomponent, variables will be highly interdependent. Between subcomponents, the interdependencies (if any) will be weak. The proposed EACC-G has been designed to decompose a nonseparable problem following the above principle. It optimizes a group of tightly interdependent variables (a subcomponent) together, rather than individually as it was done by other existing algorithms. To gain a better understanding of how such a grouping strategy used by EACC-G can help to increase the chance of capturing the interdependencies of variables, the following theorem shows a simple case of the probability to optimize two interacting variables together in EACC-G.

Theorem 1. *The probability of EACC-G to assign two interacting variables x_i and x_j into a single subcomponent for at least k cycles is:*

$$P_k = \sum_{r=k}^{N} \binom{N}{r} \left(\frac{1}{m}\right)^r \left(1 - \frac{1}{m}\right)^{N-r} \tag{3}$$

where N is the total number of cycles and m is the number of subcomponents.

Proof 1. Firstly, in each separate cycle, the probability of EACC-G to assign x_i *and* x_j into the same subcomponent is:

$$p = \frac{\binom{m}{1}}{m^2} = \frac{1}{m}.$$

In all N cycles, EACC-G will decompose the objective vector for N times, and these decomposition operations are independent of each other. Let p_r be the probability of EACC-G to assign x_i and x_j into a single subcomponent for exactly r cycles. Obviously, p_r satisfies the binomial distribution, so it is easy to get:

$$p_r = \binom{N}{r} p^r (1-p)^{N-r} = \binom{N}{r} \left(\frac{1}{m}\right)^r \left(1-\frac{1}{m}\right)^{N-r}$$

Thus,

$$P_k = \sum_{r=k}^{N} p_r = \sum_{r=k}^{N} \binom{N}{r} \left(\frac{1}{m}\right)^r \left(1-\frac{1}{m}\right)^{N-r}.$$

The theorem is proved. □

Given $n = 1000, s = 100$, we know that the number of subcomponents would be $m = n/s = 10$. If the number of cycles $N = 50$ [6], we have:

$$\begin{aligned} P_1 &= \sum_{r=1}^{N} p_r = 1 - p_0 = 1 - \left(1-\frac{1}{m}\right)^N \\ &= 1 - \left(1-\frac{1}{10}\right)^{50} = 0.9948, \\ P_2 &= \sum_{r=2}^{N} p_r = 1 - p_0 - p_1 = 0.9662. \end{aligned}$$

P_1 *and* P_2 show that EACC-G has relatively high probabilities to optimize interacting variables for at least one or two cycles. In other words, the simple grouping strategies used by EACC-G is quite effective in capturing variable interdependencies with little domain knowledge.

4. Self-adaptive differential evolution with neighbourhood search

Differential evolution (DE) [12,18] is a simple yet effective algorithm for global optimization. It has shown superior performance on benchmark functions [23] as well as real-world problems [17,21]. The primary driving force behind DE is mutation. DE executes its mutation by adding a weighted difference vector between two individuals to a third individual. Then the mutated individuals will be subject to discrete crossover and greedy selection with corresponding individuals of the last generation to produce offspring. Much work has been done to study the settings of DE's control parameters (e.g., population size *NP*, crossover rate *CR* and scaling factor *F*) [3,28]. Self-adaptive strategy has also been investigated to adapt these parameters [1] as well as different schemes [13]. DE has been combined with other algorithms to produce various hybrid algorithms [20,29]. In this section we will introduce one of the most recent DE variants as the base optimizer for subcomponents in our new CC algorithm.

4.1. Differential evolution

4.1.1. Classical differential evolution

Individuals in DE are represented by n-dimensional vectors $\mathbf{x_i}, \forall i \in \{1, \ldots, NP\}$, where NP is the population size. According to the description by Storn and Price [18], the main operations of classical DE can be summarized as follows:

(1) Mutation:

$$\mathbf{v_i} = \mathbf{x_{i_1}} + F \cdot (\mathbf{x_{i_2}} - \mathbf{x_{i_3}}) \tag{4}$$

where $i, i_1, i_2, i_3 \in [1, NP]$, are integers and mutually different, and $F > 0$, is a constant coefficient used to control the differential variation $\mathbf{d_i} = \mathbf{x_{i_2}} - \mathbf{x_{i_3}}$.

(2) Crossover:

$$\mathbf{u_i}(j) = \begin{cases} \mathbf{v_i}(j), & \text{if } U_j(0,1) < CR \text{ or } j = j_{rand} \\ \mathbf{x_i}(j), & \text{otherwise.} \end{cases} \tag{5}$$

where $U_j(0,1)$ is an uniformly distributed random number between 0 and 1, and j_{rand} is a randomly chosen index to ensure that the trial vector $\mathbf{u_i}$ does not duplicate $\mathbf{x_i}$. $CR \in (0,1)$ is the crossover rate, which is often set to 0.9.

(3) Selection:

$$\mathbf{x'_i} = \begin{cases} \mathbf{u_i}, & \text{if } f(\mathbf{u_i}) < f(\mathbf{x_i}) \\ \mathbf{x_i}, & \text{otherwise.} \end{cases} \tag{6}$$

where $\mathbf{x'_i}$ is the offspring of $\mathbf{x_i}$ for the next generation.

Although there are several variants of classical DE, the above one, which can be classified using notation $DE/rand/1/bin$, is the most often used in practice [18,13].

4.1.2. Differential evolution with neighbourhood search (NSDE)

Neighbourhood search (NS) has been shown to play a crucial role in improving evolutionary programming (EP) algorithm's performance [27]. Characteristics of several NS operators have been studied in EP [5,27]. Although DE might be similar to the evolutionary process in EP, it lacks a relevant concept of neighbourhood search. Based on a generalization of NS strategies, a neighbourhood search differential evolution (NSDE) has been proposed [24]. NSDE is the same as the DE described in Section 4.1.1 except for Eq. (4), which is replaced by the following:

$$\mathbf{v_i} = \mathbf{x_{i_1}} + \begin{cases} \mathbf{d_i} \cdot N(0.5, 0.5), & \text{if } U(0,1) < 0.5 \\ \mathbf{d_i} \cdot \delta, & \text{otherwise.} \end{cases} \tag{7}$$

where $\mathbf{d_i} = \mathbf{x_{i_2}} - \mathbf{x_{i_3}}$ is the differential variation, $N(0.5, 0.5)$ denotes a Gaussian random number with mean 0.5 and standard deviation 0.5, and δ denotes a Cauchy random variable with scale parameter $t = 1$.

The advantages of NSDE had been studied in depth [24]. Experimental results have shown that NSDE has significant advantages over classical DE on a broad range of different benchmark functions [24]. The NS operators in NSDE can improve the diversity of NSDE's search step size and population significantly without relying on any prior knowledge about the search space.

4.1.3. Self-adaptive NSDE (SaNSDE) [26]

It has been pointed out that the control parameters and learning strategies involved in DE are highly problem dependent [3,13]. It is very time-consuming to tune parameters manually for different problems. A self-adaptive DE algorithm (SaDE) [13] was recently proposed to tune parameters automatically through evolution. In SaDE, DE's two learning strategies were selected as candidates due to their good performance. Two out of three critical parameters, i.e., crossover rate CR and scaling factor F, are adaptively tuned by SaDE during evolution. The performance of SaDE was shown to be very good on the set of 25 benchmark functions provided by *CEC2005 Special Session* [19].

Comparing to NSDE, SaDE has a quite different emphasis on improving classical DE's performance. SaDE pays special attention to self-adaption of control parameters, while NSDE tries to mix search biases of different NS operators adaptively. A new algorithm, the self-adaptive NSDE (SaNSDE), combines the advantages of NSDE and SaDE together in a single algorithm [26]. SaNSDE is same as NSDE except for the following:

(1) Introducing the self-adaptive mechanism from SaDE.
(2) Following the strategy in SaDE to dynamically adapt the value of CR.
(3) Using the same self-adaptive strategy as that in SaDE to adapt the balance between Gaussian and Cauchy operators.

SaNSDE has been shown to perform significantly better than other existing algorithms on a wide range of benchmark functions [26].

4.2. SaNSDE under the new CC framework

Given SaNSDE as the base optimizer for subcomponents, it is straightforward to design the corresponding SaNSDE with cooperative coevolution, called DECC-G, by following the steps of EACC-G in Section 3. In addition, we need to specify another optimizer for the weighting strategy of DECC-G. Although SaNSDE could be used for this purpose as well, we will use the classical DE here because we want to evaluate the effectiveness of SaNSDE as a subcomponent optimizer. It will be less complex to evaluate its contribution if we apply it only in one place in the new CC framework. The pseudocode of DECC-G is given in Fig. 1.

5. Experimental studies

5.1. Experimental setup

We have chosen the well-studied domain of function optimization as the test bed for our new CC framework in order to facilitate comparisons with other work. The performance of the proposed DECC-G algorithm will be evaluated on both classical benchmark functions [27] and the new suite of test functions provided by *CEC2005 Special Session* [19]. The algorithms used for comparison include not only conventional EAs and SaNSDE, but also other CC optimization algorithms (e.g., FEPCC [6] and DECC-O [15]). In order to make the comparisons as fair as possible, the same number of fitness evaluations (FEs) will be used for all algorithms as the stopping criterion, which is set to grow in the order of $O(n)$ [6], where n is the number of dimensions (variables) of the function. Table 1 summarises different numbers of FEs for different problem sizes.

For all experiments, we set SaNSDE's population size $NP = 100$, subcomponent dimensions $s = 100$, and the number of cycles in DECC-G to be 50.

Algorithm 4.1: DECC-G$(s, cycles, FEs, wFEs)$

$pop(1:NP, 1:n) \leftarrow random\ population$
$wpop(1:NP, 1:n/s) \leftarrow define\ weight\ population$
$(best, best\_val) \leftarrow evalate(pop)$
for $i \leftarrow 1$ **to** $cycles$
do
- $index(1:n) \leftarrow randperm(n)$
- **for** $j \leftarrow 1$ **to** n/s
  - **do**
    - $l \leftarrow (j-1) * s + 1$
    - $u \leftarrow j * s$
    - $subpop \leftarrow pop(:, index(l:u))$
    - $subpop \leftarrow SaNSDE(best, subpop, FEs)$
    - $wpop(:, j) \leftarrow random\ weight$
    - $pop(:, index(l:u)) \leftarrow subpop$
    - $(best, best\_val) \leftarrow evaluate(pop)$
- $(best, best\_index) \leftarrow findbest(pop)$
- $(rand, rand\_index) \leftarrow findrand(pop)$
- $(worst, worst\_index) \leftarrow findworst(pop)$
- $pop(best\_index, :) \leftarrow DE(best, wpop, wFEs)$
- $pop(rand\_index, :) \leftarrow DE(rand, wpop, wFEs)$
- $pop(worst\_index, :) \leftarrow DE(worst, wpop, wFEs)$
- $(best, best\_val) \leftarrow evaluate(pop)$

return $(best\_val)$

Fig. 1. The pseudocode of DECC-G.

Table 1
The number of fitness evaluations for our experimental studies

# of Dim's	100	250	500	750	1000
# of FEs	5.00e+5	1.25e+06	2.50e+06	3.75e+06	5.00e+06

5.2. Results on classical benchmark functions

We first test DECC-G's performance on a set of widely used benchmark functions for numerical optimization [27]. Functions f_1–f_{13} of the set are scalable and applicable to high-dimensional optimization evaluation. Among the 13 functions, functions f_1–f_7 are unimodal functions and functions f_8–f_{13} are multimodal functions where the number of local minima increases exponentially as the problem dimension increases [5,27]. Functions f_4 and f_5 are nonseparable, while others are separable. Details of these functions can be found in appendix of [27].

The average results over 25 independent runs of our experiments are summarized in Tables 2 and 3. The results of FEPCC were taken from [6]. Symbol "–" means that the result was not given, while "INF" means that no reasonable result was found. The best mean values among all algorithms are marked in **bold face**. The *t*-test values between DECC-G and DECC-O are also given in the last column of these tables. Figs. 2 and 3 show the evolutionary processes of representative benchmark functions.

Comparing to SaNSDE, it is clear that DECC-G performed significantly better. SaNSDE's performance deteriorated rapidly as the problem dimension increased. In contrast, DECC-G was not so sensitive to the increase of problem dimensions. DECC-G gained much better results than the non-CC algorithm on high-dimensional functions, which confirmed the advantages of CC algorithms over non-CC ones for large optimization problems.

Table 2
Comparison between DECC-G and SaNSDE, FEPCC, and DECC-O on functions f_1–f_7, with dimension $D = 500$ and 1000. All results have been averaged over 25 independent runs

Test function	# of Dim's	SaNSDE mean	FEPCC mean	DECC-O mean	DECC-G mean	DECC-G/O *t*-test
f_1	500	2.41e−11	4.90e−08	2.28e−21	**6.33e−27**	−2.06e+01[a]
	1000	6.97e+00	5.40e−08	1.77e−20	**2.17e−25**	−3.56e+01[a]
f_2	500	5.27e−02	1.30e−03	3.77e−10	**5.95e−15**	−4.73e+01[a]
	1000	1.24e+00	2.60e−03	*INF*	**5.37e−14**	–
f_3	500	2.03e−08	–	2.93e−19	**6.17e−25**	−2.15e+01[a]
	1000	6.43e+01	–	8.69e−18	**3.71e−23**	−2.87e+01[a]
f_4	500	4.07e+01	9.00e−05	6.01e+01	**4.58e−05**	−1.60e+02[a]
	1000	4.99e+01	**8.50e−05**	7.92e+01	1.01e−01	−4.85e+02[a]
f_5	500	1.33e+03	–	6.64e+02	**4.92e+02**	−5.23e+00[a]
	1000	3.31e+03	–	1.48e+03	**9.87e+02**	−9.51e+00[a]
f_6	500	3.12e+02	0.00e+00	0.00e+00	0.00e+00	0.00e+00
	1000	3.93e+03	0.00e+00	0.00e+00	0.00e+00	0.00e+00
f_7	500	1.28e+00	–	1.04e+01	**1.50e−03**	−6.88e+01[a]
	1000	1.18e+01	–	2.21e+01	**8.40e−03**	−1.39e+02[a]

[a] The value of t with 24° of freedom is significant at $\alpha = 0.05$ by a two-tailed test.

Table 3
Comparison between DECC-G and DE, SaNSDE, FEPCC, and DECC-O on functions f_8–f_{13}, with dimension $D = 500$ and 1000. All results have been averaged over 25 independent runs

Test function	# of Dim's	SaNSDE mean	FEPCC mean	DECC-O mean	DECC-G Mean	DECC-G/O *t*-test
f_8	500	−201796.5	−209316.4	**−209491**	**−209491**	0.00e+00
	1000	−372991	−418622.6	**−418983**	**−418983**	0.00e+00
f_9	500	2.84e+02	1.43e−01	1.76e+01	**0.00e+00**	−2.30e+01[a]
	1000	8.69e+02	3.13e−01	3.12e+01	**3.55e−16**	−7.13e+01[a]
f_{10}	500	7.88e+00	5.70e−04	1.86e−11	**9.13e−14**	−3.88e+01[a]
	1000	1.12e+01	9.50e−04	4.39e−11	**2.22e−13**	−4.14e+01[a]
f_{11}	500	1.82e−01	2.90e−02	5.02e−16	**4.40e−16**	−3.93e+00[a]
	1000	4.80e−01	2.50e−02	2.04e−15	**1.01e−15**	−4.37e+01 ([a])
f_{12}	500	2.96e+00	–	**2.17e−25**	4.29e−21	1.75e+01[a]
	1000	8.97e+00	–	1.08e−24	**6.89e−25**	−8.45e+00[a]
f_{13}	500	1.89e+02	–	**5.04e−23**	5.34e−18	1.40e+01[a]
	1000	7.41e+02	–	**4.82e−22**	2.55e−21	1.05e+01[a]

[a] The value of t with 24 degrees of freedom is significant at $\alpha = 0.05$ by a two-tailed test.

Comparing to FEPCC, DECC-G performed significantly better on most benchmark functions. Since most of these functions are separable, these results show that DECC-G is a far more effective algorithm than FEPCC in discovering and exploiting problem structures inherent in these functions although no prior knowledge was available to the algorithms. An interesting exception is the nonseparable function f_4 with 1000 dimension, for which FEPCC performed significantly better than DECC-G. Function f_4 is the step function that is characterized by plateaus and discontinuity. The main reason why FEPCC performs better is that it adopts a greedy fitness evaluation method [6]. The method makes FEPCC move quickly from one plateau to a lower one. Without such characteristics, FEPCC would not be expected to outperform DECC-G.

In order to compare with conventional one-dimensional based CC framework, we also implemented DECC-O, which optimizes only one variable at a time. That is, each subcomponent consists of a single variable. It is clear from the results in the tables and figures that DECC-G performed significantly better than DECC-O on nonseparable functions f_4 and f_5, which is consistent with our theoretical analysis in Section 3. Even for separable functions, DECC-G still performed better than DECC-O for all but functions f_{12} and f_{13}. On closer examination, it can be seen that functions f_{12} and f_{13} are quite easy to optimize for both DECC-G and DECC-O because both algorithms could find solutions that were extremely closer to the exact optima. This characteristic weakens the effectiveness of the adaptive weighting strategy of DECC-G, because the weighting strategy adjusts a group of variables at one time, which acts like a coarse tuning. The coarse tuning is effective only when the global optimum is sufficiently far away from the current search point [27].

To evaluate the effectiveness of the proposed weighting strategy, we have compared DECC-G against DECC-G without the weighting strategy, called DECC-G-NW. The results are summarized in Table 4. The best mean values of them are marked in **bold face**. The *t*-test values between them are given in the last column of the table. It is clear from the table that DECC-G's performance is either significantly better than or comparable to DECC-G-NW's, except for a single case of f_1 with 1000

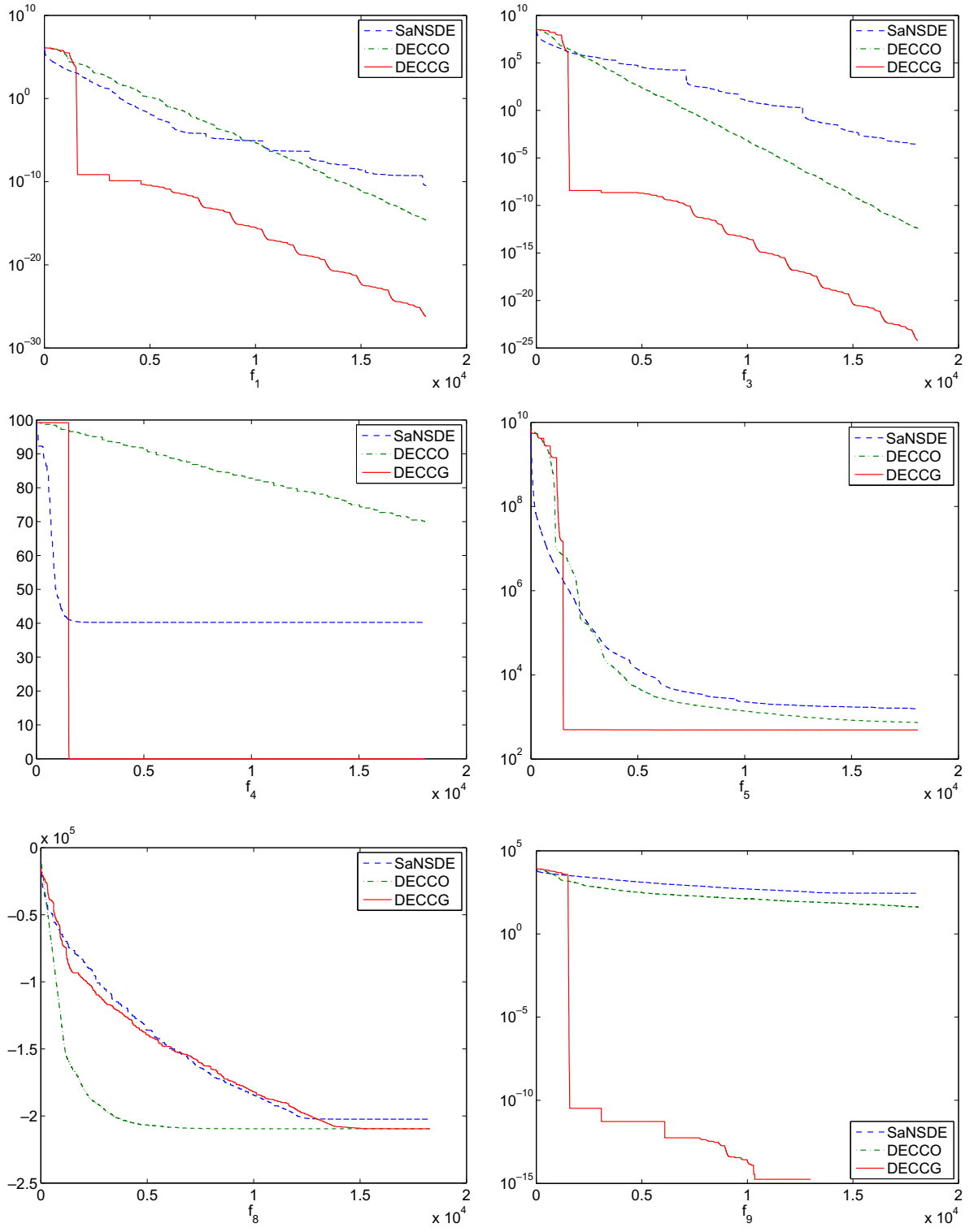

Fig. 2. The evolution process of the mean best values found for f_1, f_3, f_4, f_5, f_8 and f_9 with dimension $n = 500$. The results were averaged over 25 runs. The vertical axis is the function value and the horizontal axis is the number of generations.

dimensions. In this case, DECC-G-NW's mean performance was 2.17e−25 while DECC-G's was 2.55e−26. While there is a statistically significant difference between the two algorithms in this case, such a difference would certainly be negligible

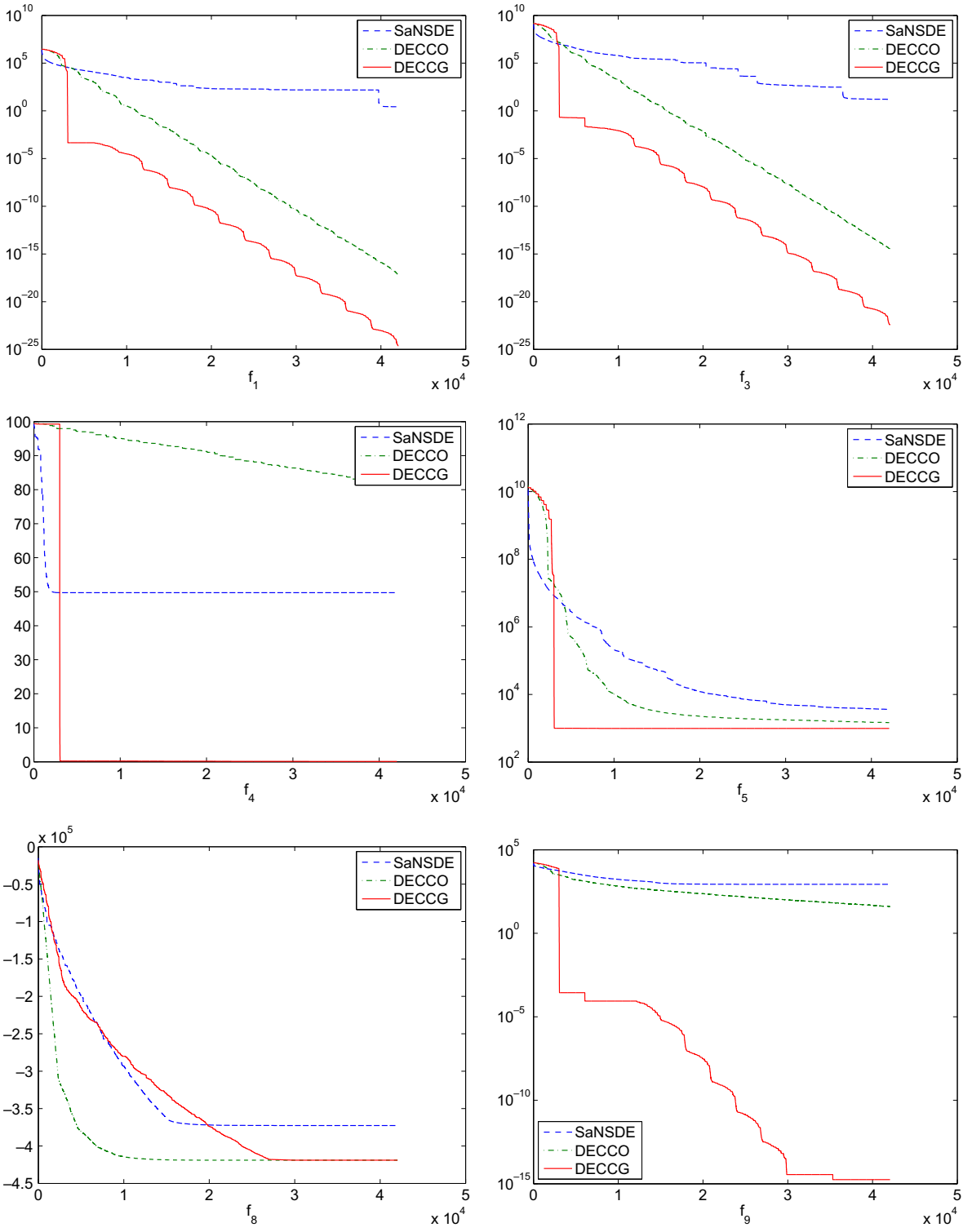

Fig. 3. The evolution process of the mean best values found for f_1, f_3, f_4, f_5, f_8 and f_9, with dimension $n = 1000$. The results were averaged over 25 runs. The vertical axis is the function value and the horizontal axis is the number of generations.

for any practical optimisation problems since both were so close to 0. In general, there is overwhelming evidence to support the effectiveness of the adaptive weighting strategy in DECC-G, especially for nonseparable functions.

Table 4
Comparison between DECC-G and DECC-G-NW (without the weighting strategy), with dimension $D = 500$ and 1000. All results have been averaged over 25 independent runs

Test function	# of Dim's	DECC-O Mean	DECC-G-NW Mean	DECC-G Mean	DECC-G/NW t-test
f_1	500	2.28e−21	1.48e−26	**6.33e−27**	−8.19e+00[a]
	1000	1.77e−20	**2.55e−26**	2.17e−25	1.65e+01[a]
f_4	500	6.01e+01	5.28e+01	**4.58e−05**	−6.78e+01[a]
	1000	7.92e+01	8.30e+01	**1.01e−01**	−1.69e+02[a]
f_5	500	6.64e+02	1.27e+03	**4.92e+02**	−1.94e+01[a]
	1000	1.48e+03	2.63e+03	**9.87e+02**	−3.05e+01[a]
f_8	500	−209491	−209491	−209491	0.00e+00
	1000	−418983	−418983	−418983	0.00e+00
f_9	500	1.76e+01	2.21e+02	**0.00e+00**	−1.17e+02[a]
	1000	3.12e+01	4.56e+02	**3.55e−16**	−1.17e+02[a]

[a] The value of t with 24 degrees of freedom is significant at $\alpha = 0.05$ by a two-tailed test.

Table 5
Comparison between DECC-G and SaNSDE and DECC-O on CEC2005 functions, with dimension $D = 500$ and 1000. All results have been averaged over 25 independent runs

CEC'05 function	# of Dim's	SaNSDE Mean	DECC-O Mean	DECC-G Mean	DECC-G/O t-test
f_{cec1}	500	2.61e−11	1.04e−12	**3.71e−13**	−2.87e+01[a]
	1000	1.17e+00	3.66e−08	**6.84e−13**	−2.96e+01[a]
f_{cec3}	500	6.88e+08	4.78e+08	**3.06e+08**	−1.13e+01[a]
	1000	2.34e+09	1.08e+09	**8.11e+08**	−1.45e+01[a]
f_{cec5}	500	4.96e+05	2.40e+05	**1.15e+05**	−7.72e+01[a]
	1000	5.03e+05	3.73e+05	**2.20e+05**	−4.12e+01[a]
f_{cec6}	500	2.71e+03	1.71e+03	**1.56e+03**	$-2.74e-01$
	1000	1.35e+04	3.13e+03	**2.22e+03**	−1.10e+01[a]
f_{cec8}	500	2.15e+01	**2.14e+01**	2.16e+01	2.72e+01[a]
	1000	2.16e+01	**2.14e+01**	2.16e+01	6.11e+01[a]
f_{cec9}	500	6.60e+02	**8.66e+00**	4.50e+02	9.09e+01[a]
	1000	3.20e+03	**8.96e+01**	6.32e+02	1.39e+02[a]
f_{cec10}	500	6.97e+03	1.50e+04	**5.33e+03**	−5.95e+01[a]
	1000	1.27e+04	3.18e+04	**9.73e+03**	−2.95e+01[a]
f_{cec13}	500	2.53e+02	**2.81e+01**	2.09e+02	7.72e+01[a]
	1000	6.61e+02	**7.52e+01**	3.56e+02	5.00e+01[a]

[a] The value of t with 24 degrees of freedom is significant at $\alpha = 0.05$ by a two-tailed test.

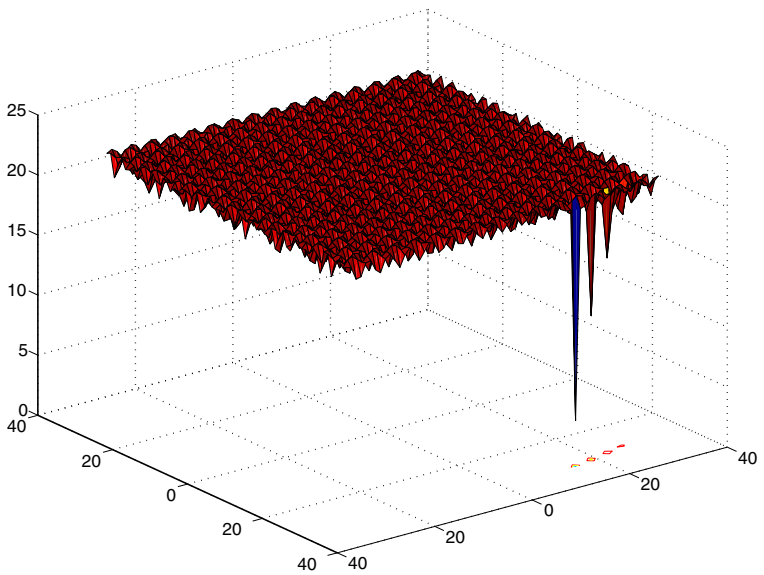

Fig. 4. The fitness landscape of two-dimensional version of f_{cec8}.

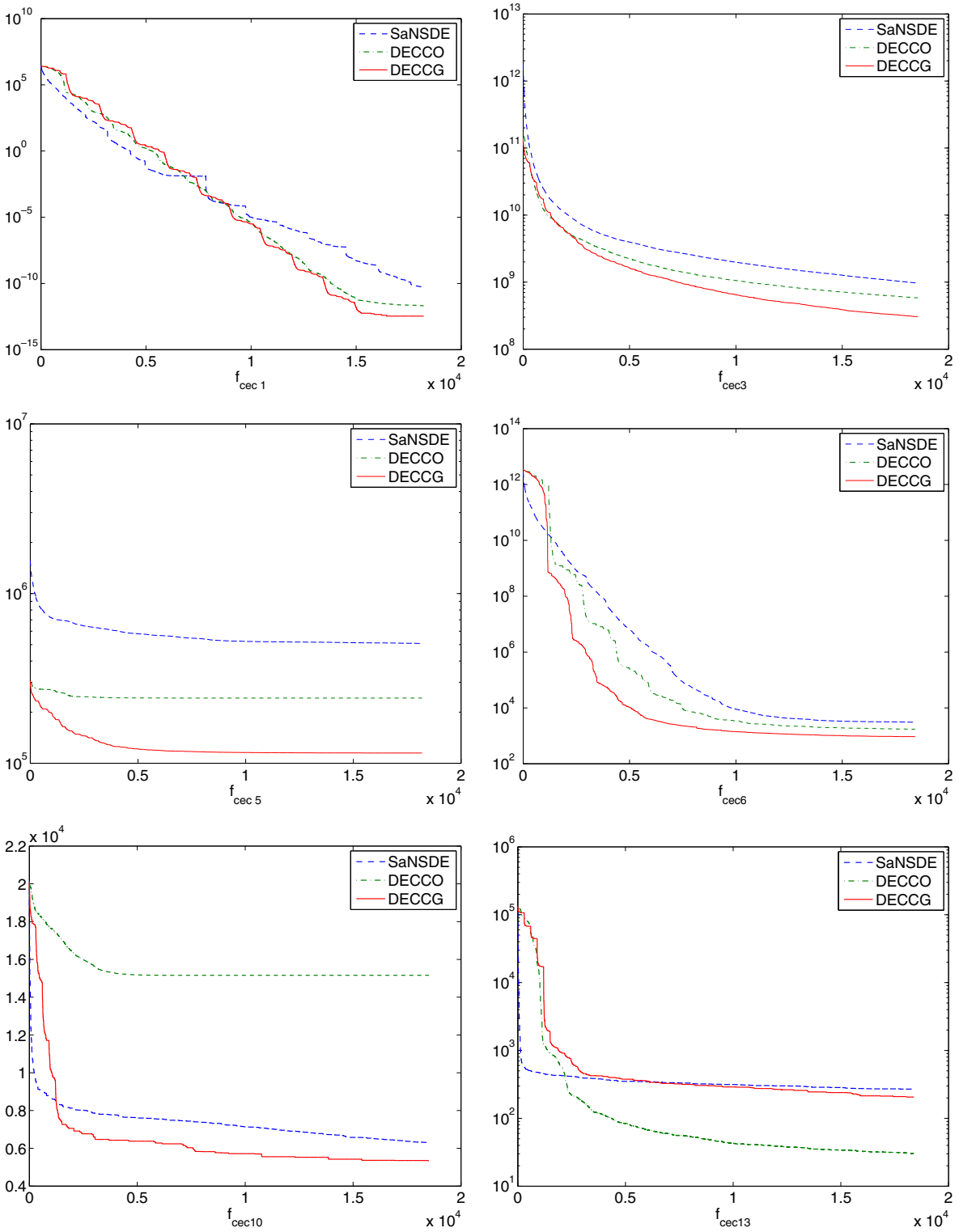

Fig. 5. The evolutionary process of the mean best values found for f_{cec1}, f_{cec3}, f_{cec5}, f_{cec6}, f_{cec10} and f_{cec13}, with dimension $n = 500$. The results were averaged over 25 runs. The vertical axis is the function value and the horizontal axis is the number of generations.

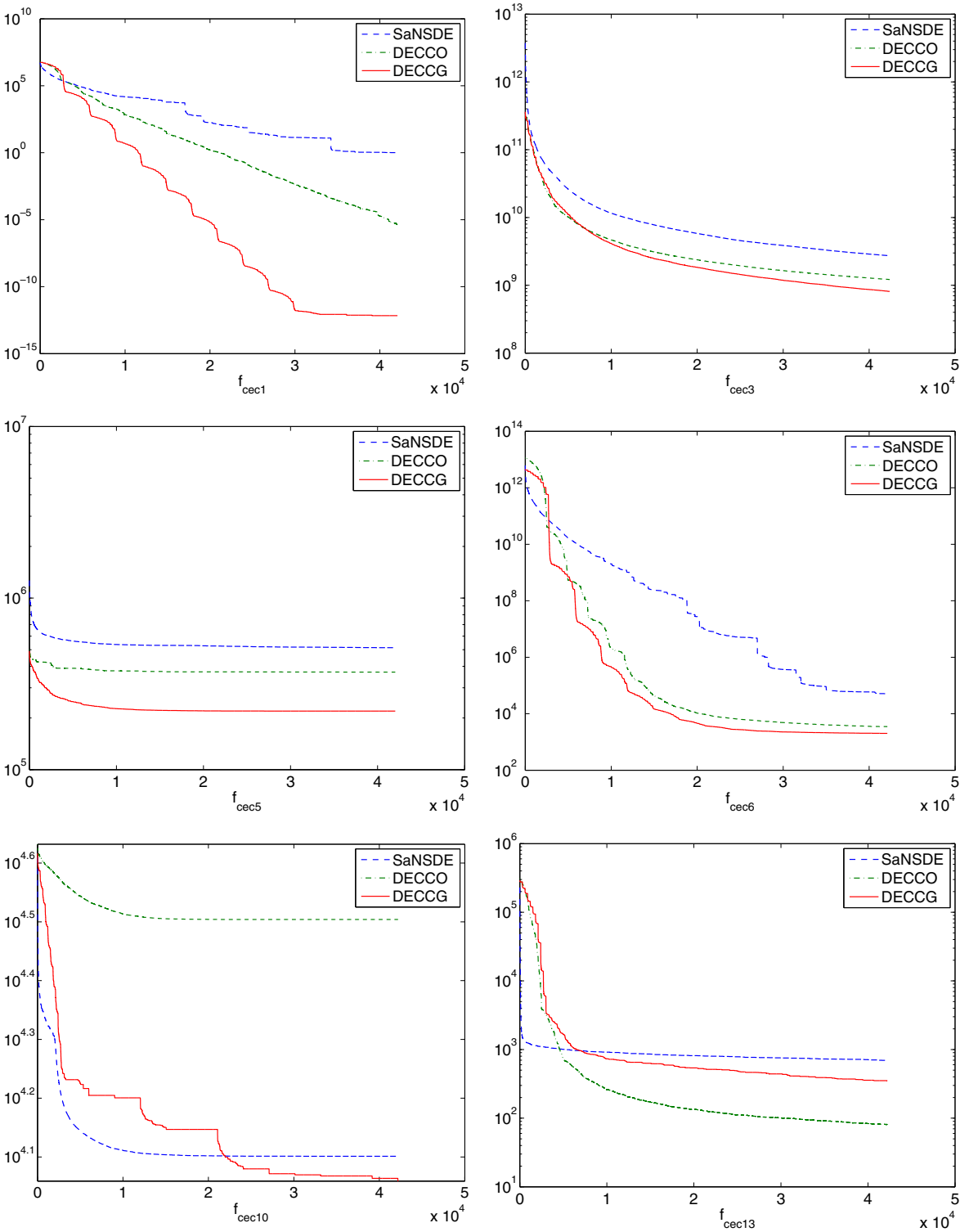

Fig. 6. The evolutionary process of the mean best values found for $f_{cec1}, f_{cec3}, f_{cec5}, f_{cec6}, f_{cec10}$ and f_{cec13}, with dimension $n = 1000$. The results were averaged over 25 runs. The vertical axis is the function value and the horizontal axis is the number of generations.

5.3. Results on CEC2005 functions

To evaluate our new CC framework further, a new set of benchmark functions, which was provided by *CEC2005 Special Session* [19], are used in this section. It includes 25 functions with varying complexity. Functions f_{cec1}—f_{cec5} are unimodal while the remaining 20 functions are multimodal. All these functions are scalable. The detailed descriptions of them can be found in [19]. Many of them are the shifted, rotated, expanded and/or combined variants of the classical functions. Some of these changes cause them to be more resistant to simple search tricks. Other changes, such as rotation, transfer separable functions into nonseparable ones, which will be particularly challenging to the CC framework. Hence, this suite of benchmark functions are ideal for experimental evaluation of CC framework based algorithms.

Since fitness evaluation of high-dimensional functions is very time consuming and the values of some functions are too big to be represented in a computer, we used only eight representative functions (out of 25) in our experimental studies, including two separable functions (f_{cec1} and f_{cec9}) and six nonseparable functions. The experimental results on these functions are given in Table 5.

It is obvious from the table that DECC-G performed better than SaNSDE for all functions except for f_{cec8}, where the performance seems to be the same for both algorithms. It is worth noting that the difference between DECC-G and SaNSDE becomes larger when the dimension increased from 500 to 1000, which shows the better scalability of DECC-G.

In comparison with DECC-O, DECC-G performed significantly better on five out of eight problems, but was outperformed by DECC-O on the remaining three Among the three functions where DECC-O performed better, f_{cec9} is what one would expect since it is a separable function. A closer look at the actual results for f_{cec8} from both algorithms revealed that they are very close. For example, their mean values were 2.16e+01 and 2.14e+01, respectively. In fact, all algorithms have similar performance on function f_{cec8}. The fitness landscape of f_{cec8} in Fig. 4 shows that it is a strange deceptive problem. Although DECC-O happened to find better local optima, they are still far way from the global optimum (with fitness value 0). Nonseparable function f_{cec13} is, in essence, the only true exception for which DECC-G was outperformed by DECC-O unexpectedly. f_{cec13} is composite function, which is composed of two kinds of different functions. This makes its characteristics hard to analyse. Our future work will focus on the analysis of DECC-G's evolutionary behaviors on functions like f_{cec13}.

To gain a better understanding of the entire search process, Figs. 5 and 6 show the evolutionary processes of six functions. (The remaining two functions were omitted to save spaces.) It can be seen from the two figures that except for function f_{cec13}, DECC-G not only obtained the best results, but also converged much faster. The effectiveness and efficiency of DECC-G is especially prominent on nonseparable functions f_{cec5}, f_{cec6} and f_{cec10}.

6. Conclusions

This paper proposes a new CC framework for high-dimensional optimization problems. It is particularly good at dealing with nonseparable problems. The key ideas behind our new CC framework include the grouping strategy and the adaptive weighting strategy. A theoretical analysis was given in the paper to illustrate how such strategies can help to capture variable interdependencies in nonseparable problems. Combined with a powerful DE algorithm—SaNSDE, we presented a novel CC optimization algorithm, DECC-G, for large problems. Extensive computational studies were carried out to evaluate the performance of DECC-G on a wide range of different benchmark functions. The results confirmed our analysis that DECC-G is very effective and efficient in tackling large optimisation problems with dimensions up to 1000.

Although we used SaNSDE as the basic subcomponent optimizer, any other EA can be introduced into our CC framework easily. Moreover, more than one kind of EAs could also be employed as the subcomponent optimizer to mix different search biases. Our future work will focus on more in-depth analysis of DECC-G and its relation to different problem characteristics. For example, it will be interesting to analyse why DECC-G did not perform as well as DECC-O on f_{cec13}. Such analysis is expected to shed light on why and how DECC-G works.

Acknowledgements

This work is partially supported by the National Natural Science Foundation of China (Grant No. 60428202), the Fund for Foreign Scholars in University Research and Teaching Programs (Grant No. B07033), and the Graduate Innovation Fund of University of Science and Technology of China (Grant No. KD2007044).

References

[1] J. Brest, S. Greiner, B. Boskovic, M. Mernik, V. Žumer, Self-adapting control parameters in differential evolution: a comparative study on numerical benchmark problems, IEEE Transactions on Evolutionary Computation 10 (6) (2006) 646–657.
[2] X. Cao, H. Qiao, J. Keane, A low-cost pedestrian-detection system with a single optical camera, IEEE Transactions on Intelligent Transportation Systems 9 (1) (2008) 58–67.
[3] R. Gamperle, S.D. Muller, P. Koumoutsakos, A parameter study for differential evolution, in: Proceedings WSEAS International Conference on Advances in Intelligent Systems, Fuzzy Systems, Evolutionary Computation, 2002, pp. 293–298.
[4] N. García-Pedrajas, C. Hervás-Martínez, J. Muñoz Pérez, COVNET: a cooperative coevolutionary model for evolving artificial neural networks, IEEE Transactions on Neural Networks 14 (3) (2003) 575–596.
[5] C.Y. Lee, X. Yao, Evolutionary programming using mutations based on the Lévy probability distribution, IEEE Transactions on Evolutionary Computation 8 (1) (2004) 1–13.

[6] Y. Liu, X. Yao, Q. Zhao, T. Higuchi, Scaling up fast evolutionary programming with cooperative coevolution, in: Proceedings of the 2001 Congress on Evolutionary Computation, 2001, pp. 1101–1108.
[7] L. Panait, S. Luke, Cooperative multi-agent learning: the state of the art, Autonomous Agents and Multi-Agent Systems 11 (3) (2005) 387–434.
[8] L. Panait, S. Luke, R. Wiegand, Biasing coevolutionary search for optimal multiagent behaviors, IEEE Transactions on Evolutionary Computation 10 (6) (2006) 629–645.
[9] M. Potter, The Design and Analysis of a Computational Model of Cooperative Coevolution, Ph.D. Thesis, George Mason University, 1997.
[10] M. Potter, K. De Jong, A cooperative coevolutionary approach to function optimization, in: Proceedings of the Third Conference on Parallel Problem Solving from Nature, vol. 2, 1994, pp. 249–257.
[11] M. Potter, K. De Jong, Cooperative coevolution: an architecture for evolving coadapted subcomponents, Evolutionary Computation 8 (1) (2000) 1–29.
[12] K. Price, R. Storn, J. Lampinen, Differential Evolution: A Practical Approach to Global Optimization, Springer, 2005, ISBN 3-540-20950-6.
[13] A.K. Qin, P.N. Suganthan, Self-adaptive differential evolution algorithm for numerical optimization, in: Proceedings of the 2005 IEEE Congress on Evolutionary Computation, vol. 2, 2005, pp. 1785–1791.
[14] R. Sarker, M. Mohammadian, X. Yao, Evolutionary Optimization, Kluwer Academic Publishers, Norwell, MA, USA, 2002.
[15] Y. Shi, H. Teng, Z. Li, Cooperative co-evolutionary differential evolution for function optimization, in: Proceedings of the First International Conference on Natural Computation, Springer-Verlag, 2005, pp. 1080–1088.
[16] D. Sofge, K. De Jong, A. Schultz, A blended population approach to cooperative coevolution for decomposition of complex problems, in: Proceedings of the 2002 Congress on Evolutionary Computation, vol. 1, 2002, pp. 413–418.
[17] R. Storn, System design by constraint adaptation and differential evolution, IEEE Transactions on Evolutionary Computation 3 (1) (1999) 22–34.
[18] R. Storn, K. Price, Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces, Journal of Global Optimization 11 (4) (1997) 341–359.
[19] P.N. Suganthan, N. Hansen, J.J. Liang, K. Deb, Y.P. Chen, A. Auger, S. Tiwari, Problem definitions and evaluation criteria for the CEC 2005 special session on real-parameter optimization, Technical Report, Nanyang Technological University, Singapore, 2005, http://www.ntu.edu.sg/home/EPNSugan.
[20] J. Sun, Q. Zhang, E. Tsang, DE/EDA: a new evolutionary algorithm for global optimization, Information Sciences 169 (3–4) (2005) 249–262.
[21] R. Thomsen, Flexible ligand docking using differential evolution, in: Proceedings of the 2003 Congress on Evolutionary Computation, vol. 4, 2003, pp. 2354–2361.
[22] F. van den Bergh, A.P. Engelbrecht, A cooperative approach to particle swarm optimization, IEEE Transactions on Evolutionary Computation 8 (3) (2004) 225–239.
[23] J. Vesterstrom, R. Thomsen, A comparative study of differential evolution, particle swarm optimization, and evolutionary algorithms on numerical benchmark problems, in: Proceedings of the 2004 Congress on Evolutionary Computation, vol. 2, 2004, pp. 1980–1987.
[24] Z. Yang, J. He, X. Yao, Making a difference to differential evolution, in: Z. Michalewicz, P. Siarry (Eds.), Advances in Metaheuristics for Hard Optimization, Springer-Verlag, 2008, pp. 397–414.
[25] Z. Yang, K. Tang, X. Yao, Differential evolution for high-dimensional function optimization, in: Proceedings of the 2007 Congress on Evolutionary Computation, 2007, pp. 3523–3530.
[26] Z. Yang, K. Tang, X. Yao, Self-adaptive differential evolution with neighborhood search, in: Proceedings of the 2008 Congress on Evolutionary Computation, in press.
[27] X. Yao, Y. Liu, G. Lin, Evolutionary programming made faster, IEEE Transactions on Evolutionary Computation 3 (2) (1999) 82–102.
[28] D. Zaharie, Critical values for the control parameters of differential evolution algorithms, in: Proceedings of the Eighth International Conference on Soft Computing, 2002, pp. 62–67.
[29] W. Zhang, X. Xie, DEPSO: Hybrid particle swarm with differential evolution operator, in: Proceedings of the 2003 IEEE International Conference on Systems, Man and Cybernetics, vol. 4, 2003, pp. 3816–3821.